TP5 – Transformation de Gabor et compression audio

La première représentation temps-fréquence d'un signal temporel est due à Gabor, physicien hongrois ayant obtenu le prix Nobel en 1971 pour l'invention de l'holographie. La *transformation de Gabor* (1946), qu'on appelle également *transformation de Fourier locale* ou *transformation de Fourier à fenêtre glissante*, est bien antérieure à la transformation en ondelettes (1984). Elle consiste à multiplier le signal par une *fenêtre glissante*, afin d'obtenir l'*analyse fréquentielle instantanée* du signal.

Bien que Gabor ait utilisé une gaussienne comme fenêtre glissante, c'est le plus souvent une « fonction créneau » qui est utilisée. La transformée de Gabor $\mathcal{G}\{s(t)\}$ d'un signal temporel $s(t)$ s'écrit alors :

$$\mathcal{G}\{s(t)\}_{(\tau,f)} = \mathcal{F}\{s(t)\, w(t-\tau)\}_{(f)}$$

où $\mathcal{F}$ désigne la transformée de Fourier et $w(t-\tau)$ la fenêtre glissante, qui peut être positionnée à un instant τ variable. Par conséquent, cette transformée est une fonction du temps τ et de la fréquence f (pour un signal temporel, on préfère généralement utiliser la fréquence, exprimée en Hertz, notés Hz, plutôt que la pulsation).

La représentation d'une transformée de Gabor se fait dans le plan (temps en abscisse, fréquence en ordonnée), sous la forme d'une pseudo-image dont le niveau de gris est proportionnel au module complexe. Une telle pseudo-image s'appelle un *sonagramme* (ou *spectrogramme sonore*) : cf. l'exemple de la figure 1, où la carte des couleurs est la carte des couleurs par défaut de Matlab (`colormap jet`). Les sonagrammes, qui constituent des « empreintes acoustiques », sont très commodes pour caractériser des voix ou des instruments de musique.

Exercice 1 : transformation de Gabor d'un enregistrement musical

Commencez par lancer le script `musique`, qui lit un fichier audio au format `WAV`. Vous pourrez bien sûr tester par la suite les autres fichiers `WAV` du répertoire `Audio`.

Écrivez la fonction `gabor`, appelée par le script `exercice_1`, qui permet de calculer et d'afficher la transformée de Gabor de cet enregistrement sonore. Deux spécificités de ce script méritent des explications :

- Si les colonnes de la matrice `TG` sont calculées à l'aide de la fonction `fft` de Matlab, alors les lignes de cette matrice correspondent aux fréquences, mais l'ordre dans lequel elles sont rangées est un peu déroutant : $f = 0, \ldots, f_{\max}$ pour la première moitié, puis $f = -f_{\max}, \ldots, 0$ pour la deuxième moitié. La commande `fftshift(TG,1)`, qui apparaît dans le script `exercice_1`, permet d'inverser verticalement les deux moitiés de la matrice `TG` (le deuxième paramètre indique selon quelle dimension l'inversion est effectuée). Après inversion, les fréquences correspondent donc à l'ordre « naturel » $f = -f_{\max}, \ldots, f_{\max}$.

- Par défaut, les axes des graphiques affichés par Matlab sont orientés vers la droite pour les abscisses, mais vers le bas pour les ordonnées (repère indirect). La commande `axis xy` permet de forcer l'orientation de l'axe des ordonnées vers le haut (la situation par défaut étant équivalente à `axis ij`).

Dans le script `exercice_1`, les décalages successifs de la fonction créneau utilisés pour le calcul de la transformation de Gabor sont choisis de manière à former une *partition du signal*. La transformation de Fourier étant inversible, cette transformation de Gabor doit donc être inversible, ce que vérifie le script `exercice_1`.

En revanche, l'inversion devient impossible si seule la partie réelle de la transformée de Gabor est conservée. Modifiez la fonction `gabor` de manière à évaluer l'impact de cette perte d'information sur le son restitué (vous pourrez également évaluer cet impact lorsque seul le module complexe de la transformée de Gabor est conservé).

Exercice 2 : calcul du sonagramme complexe

Plutôt que de perdre soit la partie imaginaire, soit la phase de la transformée de Gabor, une autre façon de condenser l'information consiste à supprimer les coefficients du spectre correspondant aux fréquences négatives, ainsi que ceux qui correspondent aux plus hautes fréquences. Cette perte d'information est elle aussi irréversible, mais la différence entre le son original et le son restitué n'est pas forcément perceptible par l'oreille humaine. En effet, l'oreille « moyenne » est sensible aux fréquences comprises entre 20 Hz (sons graves) et 20000 Hz (sons aigus), mais elle ne peut discriminer deux fréquences voisines que jusqu'à 4000 Hz environ.

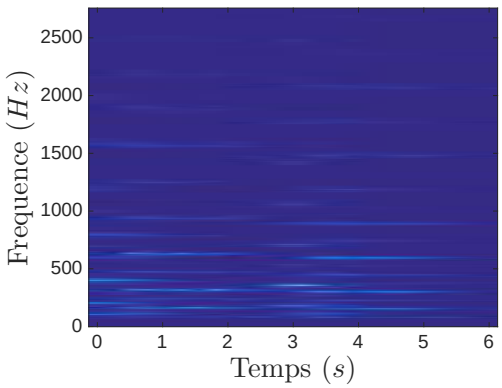

FIGURE 1 – Sonagramme d'un morceau de musique. À un instant donné et pour une fréquence donnée, le niveau de gris (affiché avec la carte des couleurs par défaut de Matlab) est proportionnel à la partie réelle du spectre.

Écrivez la fonction `sonagramme`, qui doit retourner le *sonagramme complexe* du signal original, c'est-à-dire une sous-matrice `SG` de `TG` constituée d'une partie des premières lignes de `TG`. Cette fonction permet au script `exercice_2` d'afficher le module complexe du sonagramme sous la forme d'une pseudo-image telle que celle de la figure 1, et de retourner le coefficient de compression obtenu.

Pour évaluer l'impact de la perte d'information sur le son restitué, les données manquantes de `TG` sont complétées par des zéros. Quelle est la valeur maximale du coefficient de compression pour laquelle vous jugez que le son restitué demeure fidèle à l'original ?

Exercice 3 : compression audio

La compression audio permet d'atteindre des coefficients de compression de l'ordre de 10, voire très supérieurs. Pour que le coefficient de compression du script `exercice_2` vaille 8, il faut que la proportion de fréquences positives conservées soit égale à $0,25$, auquel cas vous constatez, même en n'étant pas musicien, que le signal restitué est extrêmement dégradé. Le principe de la compression audio est légèrement différent : il consiste à ne conserver, dans le sonagramme complexe, qu'une faible proportion des coefficients les plus élevés.

Le script `exercice_3` est censé détecter, à chaque mesure, les `n` coefficients du sonagramme complexe les plus élevés (au sens du module). Écrivez la fonction `mp3` appelée par ce script, qui retourne deux matrices comportant `n` lignes : pour chaque mesure (c'est-à-dire pour chaque colonne), `indices_max` contient les numéros de lignes des `n` coefficients les plus élevés (en module), et `valeurs_max` contient les valeurs de ces coefficients.

Testez le script `exercice_3` pour différentes valeurs du paramètre `n`. Vous constatez qu'en procédant ainsi, le coefficient de compression du signal audio peut atteindre des valeurs relativement élevées, tout en garantissant une bonne restitution du son. Vous venez de réaliser un système de compression audio analogue à celui de la compression MP3 (dans une version très simplifiée).